P2PSIP
Internet-Draft
Intended status: Standards Track
Expires: August 22, 2013

J. Jimenez
Ericsson
J. Lopez-Vega
University of Granada
J. Maenpaa
G. Camarillo
Ericsson

February 18, 2013

A Constrained Application Protocol (CoAP) Usage for REsource LOcation And Discovery (RELOAD) draft-jimenez-p2psip-coap-reload-03

Abstract

This document defines a Constrained Application Protocol (CoAP) Usage for REsource LOcation And Discovery (RELOAD). The CoAP Usage provides the functionality to federate Wireless Sensor Networks (WSN) in a peer-to-peer fashion. The CoAP Usage also provides a rendezvous service for CoAP Nodes and caching of sensor information. The RELOAD AppAttach method is used to establish a direct connection between nodes through which CoAP messages are exchanged.

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) is a specialized web transfer protocol. It realizes the Representational State Transfer (REST) architecture for the most constrained nodes, such as sensors and actuators. CoAP can be used not only between nodes on the same constrained network but also between constrained nodes and nodes on the Internet. The latter is possible since CoAP can be translated to Hypertext Transfer Protocol (HTTP) for integration with the web. Application areas of CoAP include different forms of M2M communication, such as home automation, construction, health care or transportation. Areas with heavy use of sensor and actuator devices that monitor and interact with the surrounding environment.

The CoAP Usage for RELOAD allows CoAP nodes to store resources in a RELOAD peer-to-peer overlay, provides a rendezvous service, and enables the use of RELOAD overlay as a cache for sensor data. This functionality is implemented in the RELOAD overlay itself, without the use of centralized servers. The CoAP Usage involves three basic functions:

1. Registration: CoAP nodes can use the RELOAD data storage functionality to store a mapping from their CoAP URI to their Node-ID in the overlay, and to retrieve the Node-IDs of other nodes.
2. Rendezvous: Once a CoAP node has identified the Node-ID for an URI it wishes to retrieve, it can use the RELOAD message routing system to set up a direct connection which can be used to exchange CoAP messages.
3. Caching: Nodes can use the RELOAD overlay as a caching mechanism for their sensor information. This is specially useful for battery constrained nodes that can make their data available in the cache provided by the overlay while in sleep mode.

For instance, a CoAP proxy (See Section 3) could register its Node-ID (e.g. "9996172") and a list of sensors (e.g. "/temperature-1; ./temperature-2; ./temperature-3") under its URI (e.g. "coap://overlay-1.com/proxy-1/").

When a node wants to discover the values associated with that URI, it queries the overlay for "coap://overlay-1.com/proxy-1/" and gets back the Node-ID of the proxy and the list of its associated sensors. The requesting node can then use the RELOAD overlay to establish a direct connection with the proxy and to read sensor values.

Moreover, the CoAP proxy can store the sensor information in the overlay. In this way information can be retrieved directly from the overlay without performing a direct connection to the storing proxy.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

We use the terminology and definitions from Concepts and Terminology for Peer to Peer SIP [I-D.ietf-p2psip-concepts] and the RELOAD Base Protocol [I-D.ietf-p2psip-base] extensively in this document.

3. Architecture

In our architecture we extend the different nodes present in RELOAD (Peer, Client) and add support for sensor devices or other constrained devices. Figure 1 illustrates our architecture. The different nodes, according to their functionality are :

Client
: Devices that are capable of participating in a RELOAD overlay as client nodes, that is they do not route messages in the overlay.

Router
: Devices that are members of (i.e., peers in) a RELOAD overlay and capable of forwarding RELOAD messages following a path through the overlay to the destination.

Sensor
: Devices capable of measuring a physical quantity. Sensors usually acquire quantifiable information about their surrounding environment such as: temperature, humidity, electric current, moisture, radiation, and so on.

Actuator
: Devices capable of interacting and affecting their environment such as: electrical motors, pneumatic actuators, electric switches, and so on.

Proxy
: Devices having sufficient resources to run RELOAD either as client or peer. These devices are located at the edge of the sensor network and, in case of Wireless Sensor Networks (WSN), act as coordinators of the network.

Physical devices can have one or several of the previous functional roles. According to the functionalities that are present in each of the nodes, they can be:

Constrained Node

A Constrained Node (CN) is a node with limited computational capabilities. If it is wireless then it will be part of a Low-Rate Wireless Personal Area Network (LR-WPAN), it might be movable and often offer unreliable connectivity. Also, devices will usually be in sleep mode in order to prevent battery drain, and will not communicate during those periods. A CN is NOT part of the RELOAD overlay, therefore it can not act as a client, router nor proxy. A CN is always either a either a Sensor or an Actuator. In the latter case the node is often connected to a continuous energy power supply.

Reload Node

A Reload Node (RN) MUST implement the client functionality in the Overlay. Additionally the node will often be a full RELOAD peer with Proxy functionality. A RN may also be sensor or actuator since it can have those devices connected to it.

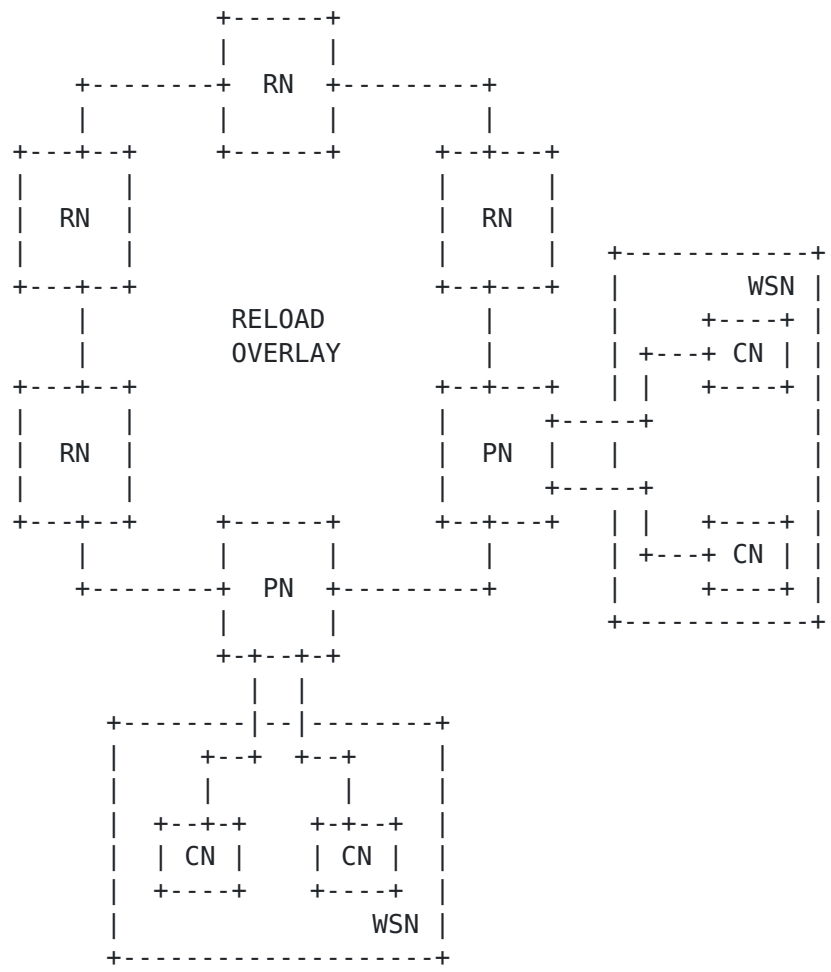

Figure 1: Architecture

4. Registering CoAP URIs

CoAP URIs are typically resolved using a DNS. When CoAP is needed in a RELOAD environment, URI resolution is provided by the overlay as a whole. Instead of registering register a URI, a peer stores a CoAPRegistration structure under a hash of its own URI. This uses the CoAP REGISTRATION Kind-ID, which is formally defined in Section 6, and that uses a DICTIONARY data model.

As an example, if a CoAP proxy that is located in an overlay overlay-1.com using a Node-ID "9996172" wants to register three different temperature sensors to the URI "coap://overlay-1.com/proxy-1/.well-known/", it might store the following mapping in the overlay:

```
Resource-ID = h(coap://overlay-1.com/proxy-1/.well-known/)
KEY =    9996172,
VALUE = {./temperature-1;
        ./temperature-2;
        ./temperature-3}
```

Note that the Resource-ID stored in the overlay is calculated as hash over the URI (i.e. h(URI)), for instance SHA-1 in RELOAD.

This would inform any other node performing a lookup for the previous URI "coap://overlay-1.com/proxy-1/.well-known" that the Node-ID value for proxy-1 is "9996172". In addition, this mapping provides relevant information as to the number of sensors (CNs) and the URI path to connect to them using CoAP.

5. Rendezvous

The RELOAD overlay supports rendezvous by fetching mapping information between CoAP URIs and Node-IDs.

As an example, if a node RN located in the overlay overlay-1.com wishes to read which resources are served at a RN with URI coap://overlay-1.com/proxy-1/, it performs a fetch in the overlay. The Resource-ID used in this fetch is a SHA-1 hash over the URI "coap://overlay-1.com/proxy-1/.well-known/".

After this fetch request, the overlay will return the following result:

```
Resource-ID = h(coap://overlay-1.com/proxy-1/.well-known/)
KEY =    9996172,
VALUE = { ./temperature-1;
          ./temperature-2;
          ./temperature-3}
```

The obtained KEY is the Node-ID of the RN responsible of this KEY/ VALUE pair. The VALUE is the set of URIs necessary to read data from the CNs associated with the RN.

Using the RELOAD DICTIONARY model allows for multiple nodes to perform a store to the same Resource-ID. This feature allows for performing rendezvous with multiple RNs that host CNs of the same class.

As an example, a fetch to the URI "coap://overlay-1.com/temperature/.well-known/" could return the following results:

```
Resource-ID = h(coap://overlay-1.com/temperature/.well-known/)
KEY =    9992323,
VALUE = { ./temperature}

KEY =    9996172,
VALUE = { ./temperature-1;
          ./temperature-2;
          ./temperature-3}

KEY =   9996173,
VALUE = { ./temp-a;
          ./temp-b}
```

6. Forming a direct connection and reading data

Once a RN (e.g., node-A) has obtained the rendezvous information for a node in the overlay (e.g., proxy-1), it can open a direct connection to that node. This is performed by sending an AppAttach request to the Node-ID obtained during the rendezvous process.

After the AppAttach negotiation, node-A can access to the values of the CNs at proxy-1 using the URIs obtained during the rendezvous. Following the example in Section 5, the URIs for accessing to the CNs at proxy-1 would be:

```
coap://overlay-1.com/proxy-1/temperature-1
coap://overlay-1.com/proxy-1/temperature-2
coap://overlay-1.com/proxy-1/temperature-3
```

Note that the ".well-known" string has been removed from the URIs, as this is only used during CNs discovery. Figure 1 shows a sample of a node reading humidity data.

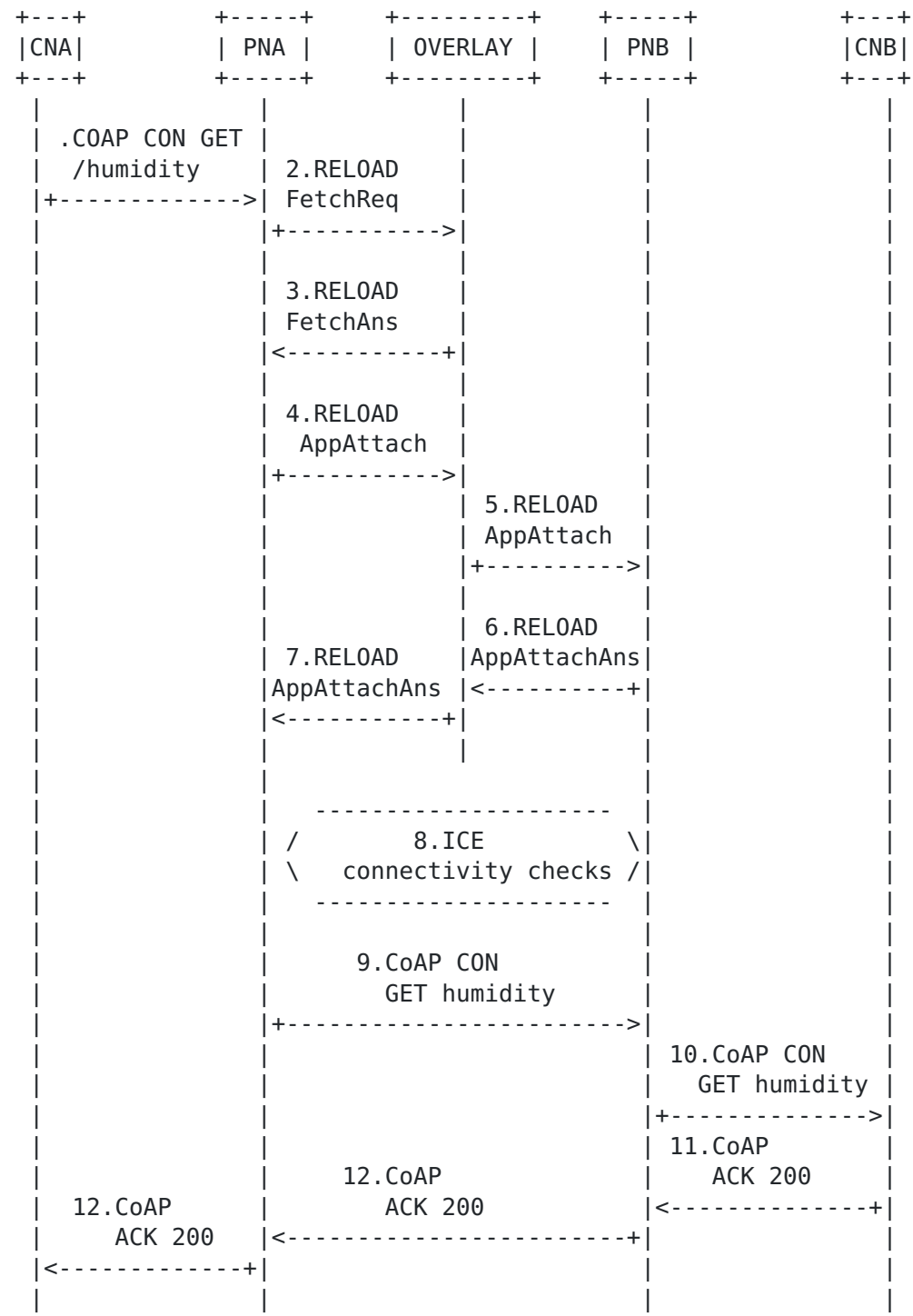

Figure 2: An Example of a Message Sequence

7. Caching Mechanisms

The CoAP protocol itself supports the caching of sensor information in order to reduce the response time and network bandwidth consumption of future, equivalent requests. This storage is done in CoAP proxies.

This CoAP usage proposes an additional caching mechanism for storing sensor information directly in the overlay. This caching mechanism is primarily intended for CNs with sensor capabilities, not for RN sensors. This is due to the battery constrains of CNs, forcing them to stay in sleep mode for long periods of time.

Whenever a CN wakes up, it sends the most recent data from its sensors to its proxy (RN), which stores the data in the overlay using a RELOAD StoredData structure defined in Section 6 of the RELOAD base draft [I-D.ietf-p2psip-base]. We use the StoredDataValue structure defined in Section 6.2 of the RELOAD base draft, in particular we use the SingleValue format type to store the cached values in the overlay. From that structure length, storage_time, lifetime and Signature are used in the same way. The only difference is DataValue which in our case can be either a ProxyCache or a SensorCache:

```
enum { reserved (0), proxy_cache(1), sensor_cache(2), (255) }
                    CoAPCachingType;
struct {
        CoAPCachingType coap_caching_type;

        select(coap_caching_type) {
                case proxy_cache:       ProxyCache proxy_cache_entry;
                case sensor_cache:      SensorCache sensor_cache_entry;

                /* extensions */

        }
} CoAPCaching;
```

7.1. ProxyCache

ProxyCache is meant to store values and sensor information (e.g. inactivity time) for all the sensors associated with a certain proxy, as well as their CoAP URIs. On the other hand, SensorCache is used for storing the information and cached value of only one sensor (CoAP URI is not necessary, as is the same as the one used for generating the Resource-ID associated to that SensorCache entry).

ProxyCache contains the fields Node-ID and series of SensorEntry types.

```
struct {
        Node-ID         Node_ID;
        uint32          length;
        SensorEntry     sensors[length];
} ProxyCache;
```

Node-ID
The Node-ID of the Proxy Node (PN) responsible for different sensor devices;
length
The length of the rest of the structure;
SensorEntry
List of sensors in the form of SensorEntry types;

SensorEntry contains the coap_uri, sensor_info and a series of SensorValue types.

```
struct {
        opaque          coap_uri;
        SensorInfo      sensor_info;
        uint32          length;
        SensorValue     sensor_value[length];
} SensorEntry;
```

coap_uri
CoAP name of the sensor device in question;
sensor_info
contains relevant sensor information;
length
The length of the rest of the structure;
sensor_value
contains a list of values stored by the sensor;

7.2. SensorCache

SensorCache: contains the information related to one sensor.

```
struct {
        Node-ID         Node_ID;
        SensorInfo      sensor_info;
        uint32          length;
        SensorValue     sensor_value[length];
} SensorCache;
```

```
   Node_ID
      identifies the Node-ID of the Proxy Node responsible for the
      sensor;
   sensor_info
      contains relevant sensor information;
   length
      The length of the rest of the structure;
   sensor_value
      contains a list of values stored by the sensor;

   SensorInfo contains relevant sensor information, such as sensor_type,
   duration_of_inactivity and c fields.

   struct {
           integer         sensor_type;
           uint32          duration_of_inactivity;
           uint32          last_awake;

           /* extensions */

   } SensorInfo;

   sensor_type
      is an integer identifying the type of the sensor.  See Figure 3;
   duration_of_inactivity
      contains the sleep pattern (if any) that the sensor device
      follows, specified in seconds;
   last_awake
      indicates the last time that the sensor was awake represented as
      the number of milliseconds elapsed since midnight Jan 1, 1970 UTC
      not counting leap seconds.  This will have the same values for
      seconds as standard UNIX time or POSIX time;

   SensorValue contains the measurement_time, lifetime and value.

   struct {
           uint32          measurement_time;
           uint32          lifetime;
           opaque          value;

           /* extensions */

   } SensorValue;
```

measurement_time
indicates the moment in which the measure was taken represented as the number of milliseconds elapsed since midnight Jan 1, 1970 UTC not counting leap seconds;
lifetime
indicates the validity time of that measured value in milliseconds since measurement_time;
value
indicates the actual value measured. It can be of different types (integer, long, string) therefore opaque has been used;

8. CoAP Usage Kinds Definition

This section defines the CoAP-REGISTRATION and CoAP-CACHING kinds.

8.1. CoAP-REGISTRATION Kind

Kind IDs
The Resource Name for the CoAP-REGISTRATION Kind-ID is the CoAP URI. The data stored is a CoAPRegistration, which contains a set of CoAP URIs.
Data Model
The data model for the CoAP-REGISTRATION Kind-ID is dictionary. The dictionary key is the Node-ID of the storing RN. This allows each RN to store a single mapping.
Access Control
URI-NODE-MATCH. The "coap:" prefix needs to be removed from the COAP URI before matching. See Section 9.

Data stored under the COAP-REGISTRATION kind is of type CoAPRegistration, defined below.

```
struct {
        Node-ID Node_ID;
        uint16 coap_uris_length;
        opaque coap_uris (0..2^16-1);
} CoAPRegistration;
```

8.2. CoAP-CACHING Kind

KindIDs
The Resource Name for the CoAP-CACHING Kind-ID is the CoAP URI. The data stored is a CoAPCaching, which contains a cached value.
Data Model
The data model for the CoAP-CACHING Kind-ID is single value.

Access Control
URI-MATCH. The "coap:" prefix needs to be removed from the COAP URI before matching. See Section 9.

Data stored under the CoAP-CACHING kind is of type CoAPCaching, defined in Section 7.

9. Access Control Rules

As specified in RELOAD base [I-D.ietf-p2psip-base], every kind which is storable in an overlay must be associated with an access control policy. This policy defines whether a request from a given node to operate on a given value should succeed or fail. Usages can define any access control rules they choose, including publicly writable values.

CoAP Usage for RELOAD requires an access control policy that allows multiple nodes in the overlay read and write access. This access is for registering and caching information using CoAP URIs as identifiers. Therefore, none of the access control policies specified in RELOAD base are sufficient [I-D.ietf-p2psip-base].

This document defines two access control policies , called URI-MATCH and URI-NODE-MATCH. In URI-MATCH policy, a given value MUST be written and overwritten if and only if the signer's certificate has an associated URI which canonicalized form hashes (using the hash function for the overlay) to the Resource-ID for the resource.

In URI-NODE-MATCH policy, a given value MUST be written and overwritten if and only if the signer's certificate has an associated URI which canonicalized form hashes (using the hash function for the overlay) to the Resource-ID for the resource. In addition, the dictionary key MUST be equal to the Node-ID in the certificate and that Node-ID MUST be the one indicated in the SignerIdentity value cert_hash.

These Access Control Policies are specified for IANA in Section Section 11.4.

10. Security Considerations

TBD.

11. IANA Considerations

11.1. RELOAD Sensor Type Registry

IANA SHALL create a "RELOAD sensor type" Registry. Entries in this registry are 16-bit integers denoting method codes as described in Section 7. The initial contents of this registry are:

Code Name	Value
temperature	0
humidity	1
acceleration	2
pressure	3
altitude	4
luminance	5
velocity	6
signal_strength	7
battery	8
heart_rate	9

Figure 3

11.2. CoAP-REGISTRATION Kind-ID

This document introduces one additional data Kind-ID to the "RELOAD Data Kind-ID" Registry:

Kind	Kind-ID	RFC
CoAP-REGISTRATION	105	RFC-AAAA

This Kind-ID was defined in Section 4.

11.3. CoAP-CACHING Kind-ID

This document introduces one additional data Kind-ID to the "RELOAD Data Kind-ID" Registry:

Kind	Kind-ID	RFC
CoAP-CACHING	106	RFC-AAAA

This Kind-ID was defined in Section 4.

11.4. Access Control Policies

IANA SHALL create a "CoAP Usage for RELOAD Access Control Policy" Registry. Entries in this registry are strings denoting access control policies, as described in Section 8.1. New entries in this registry SHALL be registered via RFC 5226 [RFC5226]. Standards Action. The initial contents of this registry are:

Access Policy	RFC
URI-NODE-MATCH	RFC-AAAA
URI-MATCH	RFC-AAAA

This access control policy was described in Section 9.

12. References

12.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-13 (work in progress), December 2012.

[I-D.ietf-p2psip-concepts]
Bryan, D., Willis, D., Shim, E., Matthews, P., and S. Dawkins, "Concepts and Terminology for Peer to Peer SIP", draft-ietf-p2psip-concepts-04 (work in progress), October 2011.

[I-D.ietf-p2psip-base]
Jennings, C., Lowekamp, B., Rescorla, E., Baset, S., and H. Schulzrinne, "REsource LOcation And Discovery (RELOAD)

Base Protocol", draft-ietf-p2psip-base-24 (work in progress), January 2013.

12.2. Informative References

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

Authors' Addresses

Jaime Jimenez
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: jaime.j.jimenez@ericsson.com

Jose M. Lopez-Vega
University of Granada
CITIC-UGR Periodista Rafael Gomez Montero 2
Granada 18071
Spain

Email: jmlvega@ugr.es

Jouni Maenpaa
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: jouni.maenpaa@ericsson.com

Gonzalo Camarillo
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: gonzalo.camarillo@ericsson.com